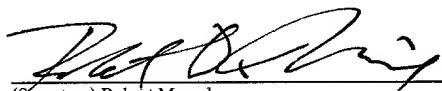# UNITED STATES PATENT APPLICATION FOR

APPLICATION MESSAGING SYSTEM WITH
FLEXIBLE MESSAGE HEADER STRUCTURE

Inventors: Shean-Guang Chang
John Wells
Stephen Felts

APPLICATION MESSAGING SYSTEM WITH FLEXIBLE MESSAGE HEADER STRUCTURE

Inventors: Shean-Guang Chang
John Wells
Stephen Felts

Copyright Notice

[0002]   This application claims priority from provisional application "APPLICATION MESSAGING SYSTEM WITH FLEXIBLE MESSAGE HEADER STRUCTURE", Application No. 60/290,197, filed May 11, 2001, and incorporated herein by reference.

[0003]   This application is related to U.S. Patent 5,265,250, filed March 29, 1990, and issued November 23, 1993, which is incorporated herein by reference.

Field of the Invention:

[0004]   The invention relates generally to messaging systems for use with distributed application servers, and particularly to a flexible message header structure for communicating messages.

Background of the Invention:

[0005] In a distributed application system, such as those provided by the Tuxedo system developed by BEA Systems, Inc., San Jose, California, most of the distributed applications use tightly coupled messages to communicate. Adding or modifying message-related application features almost always requires changing the message format, which has an impact on the rest of the software code that is exposed to the message. The change in message compatibility is an important issue in a multi-release environment.

[0006] BEA Tuxedo is just one example of such a messaging system. Earlier versions of the product have a fixed message format to support all of the existing features at the time. The message format is exposed to almost the entire Tuxedo code base. In the past small changes to the message format were made by using a few reserved fields in the message. Major additions to the messaging format were simply not allowed. One resultant problem with this approach is that a single fixed size message format makes use of the system resources and the network bandwidth in a less than efficient way.

[0007] The future growth of application server products such as Tuxedo requires an improved messaging system that is flexible enough to facilitate adding major new features with little or no impact to the existing code base. The messaging system should be efficient enough to reduce the message footprint for any features depending on it, without rewriting the code, and to be flexible enough to allow multi-release deployment without patching the older releases. Particularly, a solution to the problem as it affects Tuxedo requires a new message header in order to carry a longer Tuxedo service name and to reduce overhead both on the queues, and over the network. In addition, future projects related to application servers including Tuxedo, require a more flexible format than the existing message header so that different types of data can be carried as part of the header without implying any dependency between the different types of data in order to access,

manage or identify each data type. Finally, overall throughput is an important design criteria that should be balanced against the functionality of the messaging system.

5 ## **Summary of the Invention:**

[0008]   The Flexible Message Header (FMH) of the invention is a new typed modular message structure designed to be compact for efficiency, versatile for plug-and-play complex data type, simple for easy access, and clean for interoperability.

10 [0009]   The Typed Container Module (hereinafter referred to as the TCM) is a stateful message module with a compact header (Typed Container Header, hereinafter referred to as the TCH), a user definable payload (Typed Container Body, hereinafter referred to as the TCB) and a set of payload-specific callback to handle the lifecycle of the message module at the

15 different state such as creation, preparation, manipulation, resizing, transferring, and deletion. All of the implementation of the TCM is total hidden behind a set of abstract interfaces such that changing the implementation of the TCM has no impact on the code that is using it.

[0010]   In accordance with the invention, a single message has a small

20 generic header with various numbers of TCMs. The relationship between TCM and message is clearly defined such that the integrity of the entire message is not affected by dynamically modifying, adding or removing a TCM from the message. This facilitates the handling of the message content with great flexibility and efficiency at one end, and provides a very simple and

25 efficient way to add new features at the other end. Major message content change can be achieved simply by either adding a TCM, or swapping the TCM with a different TCM. Minor message content change can be achieved simply by either

modifying or expanding the TCM directly within the message. New features can be added simply by adding new TCM and new code to process the new TCM.

[0011]   The TCM solution provides several advantages such as the following:

[0012]   New features can be added by adding new code and new TCM without changing the existing code.

[0013]   The invention facilitates message content change with flexibility and efficiency. Old and new releases of an application server that uses messaging can be mixed in a single deployment without patching the old release.

[0014]   Reduction in message footprint by allowing each message to tailor its format to its exact need, instead of having a large common message format for all-purpose use.

[0015]   Increased performance for forwarding messages by allowing the system to examine each message piece by piece.

[0016]   Messages can be dynamically accessed at the TCM level directly. This provides flexibility and efficiency to the code that needs to access the message. By accessing message at the TCM level the system allows the TCM to be prepared for local access (e.g. decoding, decrypting, and expansion) at different levels and at different places by design, thereby enhancing performance and transparency.

[0017]   Commonly used or mandatory TCM can be prepared at a very low level in the system so that the upper level does not need to deal with it. Special or rare TCM can be passed up to the upper level without preparation, which can then decide which step to take on the TCM.

[0018]   For processes that are forwarding the message, the ability to access messages at the TCM level provides transparency by allowing the

system to examine required TCM only, and without being affected by a message having new TCM that is irrelevant to the process.

[0019]	In a multi-release environment a message using TCM can pass through an older release which may not know all the TCMs in the message, and it can forward the message along to its final destination which has the proper code to process the message. This is very useful in a very large or stable deployment by allowing incremental upgrade within a release or selectively upgrading only part of a release.

Brief Description of the Figures:

[0020]	**Figure 1** shows the transfer of a message using a Flexible Message Header structure in accordance with an embodiment of the invention.

[0021]	**Figure 2** shows a detailed view of an embodiment of the Flexible Message Header structure of Figure 1.

[0022]	**Figure 3** shows a detailed view of the pointer linking of Figure 1.

[0023]	**Figure 4** is a flowchart of a messaging process in accordance with an embodiment of the invention.

Detailed Description:

[0024]	The Flexible Message Header (FMH) is a typed modular structure that supports a series of Typed Container Modules (TCM). Each TCM is a stateful message module with a compact header (Typed Container Header, TCH), a user definable payload (Typed Container Body, TCB), and a set of payload-specific callback to handle the lifecycle of the module at the different state such as creation, preparation, manipulation, resizing, transferring, and deletion.

[0025]	In accordance with one embodiment of the invention, a single message has a small generic header, with various numbers of TCMs. The

relationship between TCM and message is clearly defined such that the integrity of the entire message is not affected by dynamically modifying, adding or removing TCM from the message at design. This can facilitate and minimize the overhead of handling the message content with great flexibility

5   and efficiency at one end and provide a very simple and efficient way to add new features at the other end.

[0026]   Major message content change can be achieved simply by either adding a TCM or by swapping the TCM with a different TCM. Minor message content change can be achieved simply by either modifying or expanding the

10   TCM directly within the message. New features can be added simply by adding a new TCM and new code to process the new TCM. Messages can be dynamically accessed at the TCM level directly. This provides a flexibility and efficiency to the code that needs to access the message. By accessing message at the TCM level the system allows TCM to be prepared for local

15   access (e.g. decoding, decrypting, and expansion) at different levels, and at different places thereby enhancing performance and transparency.

[0027]   The actual (detailed bit by bit) content of the message header itself is not necessary for understanding the invention, although to show how servers such as Tuxedo can benefit from this design, a simple demo relating

20   to Tuxedo is given to show the advantage of the design.

[0028]   It will be evident to one skilled in the art that, while Tuxedo is referred to herein for purposes of illustration, the invention is not limited to this one particular brand of application server but maybe used with any messaging system or application server that uses messaging and should be afforded the

25   broadest scope.

Definitions and Acronyms

Flexible Message Header (FMH) - A modular message structure consisting of a Flexible Message Header and multiple TCMs.

STORAGE Mode - One of the data representations of the FMH. This mode is suitable for queuing or sending over the network.

EDIT Mode - One of the data representations of the FMH. This mode is suitable for accessing or editing.

5 Typed Container Module (TCM) - The modular unit of the Flexible Message Header.

Typed Container Header (TCH) - The header of the Typed Container Module.

Typed Container Body (TCB) - The Body of the Typed Container Module.

TCM Table - A global table having all the known TCM types and their

10 associated typed buffer type and subtype. A portion of the table can be compiled in, and other portions can be added by the API if TCM registration is allowed.

TCM Pointer Set - A pointer pair that is part of the FMH and also is a prefix to the TCM when the FMH is in the EDIT mode.

15

[0029] To take the example of Tuxedo the existing Tuxedo Message Header is a 360 byte fixed format data structure which is not adaptive to the needs of the future Tuxedo releases. In addition, there are fields which are out-dated or fields which need to be re-structured to obtain better performance

20 or support new features. In order to accomplish the above tasks and extend the usefulness of the Tuxedo Message, a new generation of the Tuxedo Message Header designed in accordance with the invention can address these issues.

25 **FMH Modulation**

[0030] The FMH is a modular-based message header design with typed buffer function support and unique identification. By modularizing the Flexible Message Header into functionally related groups, it can facilitate protocol interoperability, multi-platform support, functional plug-in and direct data

manipulation at the module level.

[0031]   Each module has a type to associate it with its functionality and a specific typed buffer entry to manage the module content at various phases of the message lifecycle. Each module type has a unique TCM table entry

5   which has an index into the typed buffer table to assist with data creation, preparation, flattening and platform conversion.

[0032]   Module swapping allows messages to flow between different versions of the application server (for example) Tuxedo implementation. By converting data from one module type to another messages can be

10   transferred between different releases of Tuxedo implementations. Also, by encapsulating data inside, a typed module allows data to be examined only at the module level. This means that new data in the new module can be transparent to the old Tuxedo implementation. This feature can be used to allow messages to carry data to be received by multiple recipients who have

15   different levels of knowledge of the message content.

[0033]   New modules can be introduced without affecting the rest of the message header. Similarly, existing module can be removed. Modules can also be manipulated, such as by decoding or decrypting, independently from the rest of the message content.

20   **[0034]**   Modulation can reduce the footprint of the message header by partitioning the message header into functional modules and only transferring those modules which are needed by the recipient. Modulation with a unique identification for each module can support messages with multiple TCMs having the same type at the same time, or during the entire message lifecycle.

25   Assigning a unique identification for each module. This allows inter-module correlation.

Typed Container Module (TCM)

[0035]   The basic unit of the Flexible Message Header comprises a

header part called a Typed Container Header (TCH) and a body part called Typed Container Body (TCB). When the FMH is in EDIT mode, each TCM, except the one containing the user data, is prefixed with a TCM Attachment Unit which comprises two pointers, -- one to point to the next TCM and the other to point to the previous TCM in the message, together with a variable to record the allocated size of the TCM. The TCM containing the user data (the User TCM) does not have the TCM Attachment Unit and is instead directly attached to the FMH. When the FMH converts back into the STORAGE mode from the EDIT mode, the TCM Attachment Unit is removed from the TCMs. Listing 1 shows an example of a TCM structure in accordance with an embodiment of the invention.

```
typedef sruct _tmtcm_t TCM;
struct _tmtcm_t {
unsigned short tmtcm_type;
unsigned short tmtcm_id;
TM32U tmtcm_len;
TM32U tmtcm_flag;
TM32I tmtcm_udata; /* the beginning of the TCB */
}
typedef struct _tmtcmtbl_t TMTCMTBL
struct tmtcmtbl_t {
        int count;
        int tcmhash[TCM_MODULE + 1];
};
typedef struct tmtcm_ele_t TMTCMELE;
static struct _tmtcm_ele_t {
unsigned short tmtcm_type;
char tbtype[TMTYPELEN];
char tbstype[TMSTYPELEN];
TM32I flag; /* 0, reserved for future use */
int next_ele; /* next TCM element in the free or TCM hash table
*/
}
```

Listing 1

Typed Container Header (TCH)

[0036] The header part of the TCM comprises a type field, ID field, length field, and a flag field with Header Extension Bits (HEB), wherein these variables are defined as follows:

5　Type field - An index into a global TCM table.

ID field - A unique identification number during the message lifecycle.

Length field - The size of the TCH plus the size of the used data space in the TCB. The actual size of the allocated space for the TCM can be larger than the length field.

10　Flag field - The flags currently used are:

     TCM_IN_USED - The TCM is in use.

     TCB_ENCODE - The TCB is in "encoded" state.

     TCM_UNKNOWN - The TCM is in an unknown state.

     TCM_TTL - A flag indicating the TCM's time-to-live.

15　Header Extension Bits - The lower four bits of the flag field. The Header Extension Bits (HEB) are used to extend the TCH by increments of four bytes. 32 bit unsigned variables (which in the code examples referred to as TM32U variables; 32 bit signed integers are referred to as TM32I variables) are used so that the TCH can be decoded/encoded without the knowledge of its content. If the HEB is not zero then the lower four bits of the last four bytes in

20　the extended TCH has to be the same as the HEB. This allows traversing from the TCB back to the TCH.

[0037] The TCM_TTL flag indicates the TCM's time-to-live. The TTL flag is particularly useful when a message from a first process or application

25　to a second process or application passes through, or is routed by, an intermediate process. The intermediate process likely does not care about the content of the message itself, but may care about whether it should or should not store the message, or any portion of the message, in its memory space. The invention allows individual TCM's to have different TTL values.

Intermediate processes can use this TTL flag as an indicator of whether or not that particular TCM has changed since it was last sent. If the TTL flag indicates the TCM has not changed (for example setting the TTL flag to zero), then the memory allocated to this particular TCM can be freed for other uses.

5

Typed Container Body (TCB)

[0038] This is the data portion of the TCM, and can be any binary data format. A typed buffer switch via the TCM type is used to administrate the TCB during the message lifecycle. Accessing the TCB is done in two steps - the first step is an API call to locate the TCM containing the TCB; the second step allows direct accessing or editing of the TCB content.

Flexible Message Header (FMH)

[0039] The Flexible Message Header in accordance with the invention comprises a message header and one or more TCMs. In one embodiment the FMH has two modes; STORAGE mode and EDIT mode.

[0040] In STORAGE mode, the FMH is suitable for queuing messages or shipping them on the network. In STORAGE mode, the message is stored in one continuous memory space and all of the TCMs are stored without their TCM pointer set.

[0041] In EDIT mode, the FMH is suitable for accessing or editing. In EDIT mode, the message may be scattered over several memory spaces. The Flexible Message Header and the User TCM can be contained in one continuous memory space, and the rest of TCMs with their TCM pointer set contained in one or more other continuous memory spaces. Listing 2 shows an example of a Flexible Message Header structure.

```
typedef struct _tmFMH_t FMH
struct _tmFMH_t {
char *tcmp_f;    /* point to the tcmp_f of the  next TCM */
char *tcmp_p;    /* always to be NULL */
```

```
TM32I bsize;      /* memory size of the allocated space */
METAHDR mhdr;     /* the Tuxedo meta header */
TM32I magic;      /* magic number for the memory space allocation
*/
unsigned short tmfmh_flag;
unsigned short  tmfmh_maxid;      /* the maximum id in the
message */
}
```

Listing 2

[0042]   The FMH comprises a TCM attachment unit, a meta header, a flag field, and a maximum ID field, wherein these fields are defined as follows: TCM Attachment Unit - The TCM Attachment Unit includes two pointers to both the next and the previous TCMs respectively. It is used only when the message is in the EDIT mode, and also includes a size variable to indicate the allocated memory space of the FMH. The TCM Attachment Unit is not sent as part of the sent message.

Meta header - A 32 byte field included as part of the Message Header, and particularly the Tuxedo Message Header when the application server is Tuxedo.

Magic - a magic number included as part of the Tuxedo Message Header when the application server is Tuxedo.

Flag (unsigned short) - Indicates the status of the FMH. The flags currently used are:

FMH_IN_USED - The message is in use.

FMH_EDIT - The message is in EDIT mode.

FMH_STORAGE - The message is in STORAGE mode.

Maximum id (unsigned short) - the maximum id field denotes the largest identifier ever assigned to any TCM within the message. It increments by one every time a new TCM is added to the message. In one embodiment the maximum number for this field is 65535. It is used to assure a unique id is associated to each TCM within the message.

[0043] **Figure 1** illustrates the various phases in which an original message can be generated, modulated, and stored at a recipient's location. As shown in **Figure 1**, the message may pass through a number of phases in its transmission from one application to another. At a first application, a message **102** is initially created comprising a header information portion **104**, a body information portion **106**, and a user data portion **108**. During an editing/modulating phase a Flexible Message Header (FMH) **110** is created, together with a number of Typed Container Modules (TCM) **111**, **115**, **119**, **123**, each of which has a Typed Container Header (TCH) portion **112**, **116**, **120**, **124** and one of either a Typed Container Body (TCB) portion **114**, **118**, **122**, or a user data portion **126**. The TCM's contain the full content of the original header information, body information and user data, divided between a number of TCM's. An Attachment Unit **113**, **117**, **121** is added to each TCM that contains TCB information (in this example **111**, **115**, **119**). The Attachment Unit provides pointers to both the FMH (indicated by the arrow **128**), and to the previous and successive TCM's in the chain (indicated by arrows **130**, **132**). A TCM containing no TCB information and instead only user data (for example TCM **123**) is not prefixed with an Attachment Unit and is instead directly linked to the FMH via an independent pointer (indicated by the arrow **134**). When the message is received at the final destination and stored, the second (receiving) application retrieves only those TCM's for which it is configured to received. Other TCM's may be ignored. So, for example, the second application may choose to receive the entire message as originally sent, including the full header **104**, body **106**, and user data **108**; or it may choose to receive only header portion A **111**, Header portion B, **115**, Body portion A **114**, Body portion B, **118**; or it may choose to receive only header portion A **111**, Body portion A **114**, and user data **126**. Owing to the flexibility of the Flexible Message Header system, the application receiving the message can be configured to only listen for, receive and use a portion of the

transmitted message. This allows for less network traffic at the receiving end, and greater flexibility in modifying the messaging format at a later point in time, for example, for a new release of the application server, without having to re-write existing application code.

5   **[0044]**   **Figure 2** illustrates a detailed view of a non-user data TCM module during the edit/modulation phase. Each TCM module (not including the user data modules) includes a TCH header portion **204**, and a TCB body portion **206**. The header portion **204** comprises a number of fields that are used to describe or type that particular module, and to distinguish that module
10   from others in the same message. These fields may include, in one embodiment, a Type field **212** which can be an index into a global TCM table to denote the type of TCM; an ID field **214** for uniquely identifying the TCM during the message lifecycle; a length field **216** to denote the size of the TCH plus the size of the used data space in the TCB; and a flag field **218**, wherein
15   the flags may be any of (for example): TCM_IN_USED - the TCM is in use; TCB_ENCODE - the TCB is in "encoded" state; TCM_UNKNOWN - the TCM is in an unknown state.

[0045]   **Figure 3** shows how TCM's are linked to one another and to the FMH during the edit/modulation phase. A TCM Attachment Unit is prefixed to
20   each TCM, other than those TCM's carrying only user data. User data TCM's are linked directly to the FMH as shown in **Figure 1**, and are not shown here for reasons of clarity. TCM's which contain a TCH and a TCB are linked through their Attachment Units, which contain pointers to the preceding TCM (or the FMH) and to the succeeding TCM. In the example given in **Figure 3**,
25   a first TCM **304** is prefixed with a TCM Attachment Unit **312**, which in turn contains a Last Unit pointer **314** to the FMH (indicated by the arrow **318**), and a Next Unit pointer **316** to the succeeding (second) TCM **308** (indicated by the arrow **320**). The second TCM **308** is prefixed with a TCM Attachment Unit **324**, which in turn contains a Last Unit pointer **326** to the FMH (indicated by

the arrow **320**), and a Next Unit pointer **328** to the next TCM (indicated by the arrow **322**). The process continues for all TCM's comprising this particular message.

[0046] The method used by an embodiment of the invention in messaging between applications is shown in **Figure 4**. In step **402** a first application wishes to send a message to a second application. In step **404**, a message is generated by the first application, together with header information, body information, and user data. In step **406**, the header information and body information is segmented or modulated into header modules, and the user data is placed in a user data module. The Flexible Message Header is then created (step **408**), and in step **410** an attachment unit added to each container module so that the container modules can be linked to the Flexible Message Header and to each subsequent container module in succession. The user data module is independently linked to the Flexible Message Header (step **412**). Modulated as such the message can be sent as a series of typed container modules to the recipient applications (step **414**), where in step **416** a second application can receive them, and selecting among certain of the container modules, reconstruct and store the message according to its particular configuration needs.

Function Interfaces (API's)

[0047] The following section details various FMH-related functions that may be used by a developer in accordance with the invention to manipulate or access the TCM, TCB, and FMH. The particular functions listed here illustrate one embodiment of the invention. A detailed description of other API functions that can be used with the invention, including those not specifically related to the FMH header structure (such as the _tminitbuf(), and _tmreinitbuf() functions), are given in the related U.S. Patent 5,265,250, incorporated by reference, which also contains additional detail on the

functions described herein as they pertain to non-FMH systems. It will be evident to those skilled in the art that equivalent or alternative function calls can be used to effect similar functionality, and are within the spirit and scope of the invention.

5

```
#if defined(__STRICT_ANSI)
typedef TCBP void *
#else
typedef TCBP char *
#endif
```

10

Listing 3

Register TCM Table entry (_tmfmsg_regtcm)

```
int _tmfmsg_regtcm(unsigned short tcmtype, char *type, char *subtype,
TM32U flag)
```

15

[0048]   This function creates a TCM Table entry with the tcmtype as the unique key and its associated typed buffer type and subtype. When searching the corresponding typed buffer table entry, an Internal Typed Buffer Table is used first, then a User Typed Buffer Table. The search stops at the first match of both the type and the subtype. Upon successful completion, a new TCM Table entry is generated.

20

Deregister TCM table entry (_tmfmsg_dregtcm)

```
int _tmfmsg_dregtcm(unsigned short tcmtype, TM32U flag)
```

25

[0049]   This function removes an existing TCM Table entry matching the tcmtype. Upon successful completion, the matching entry is removed from the TCM Table.

30

Locate TCM (_tmfmsg_loctcm)

```
int _tmfmsg_loctcm(unsigned short tcmtype, TM32U flag)
```

[0050]   This function returns the results of the search for the specified
tcmtype entry in the TCM table. Upon successful completion, the function
indicates that the given tcmtype is existing in the TCM table.

Create TFMM Object (_tmfmsg_create)

```
FMH * _tmfmsg_create(char *type, char *subtype, TM32U len, unsigned
short *id, TM32U flag)
```

[0051]   This function generates a FMH object, and is typically the first
function called during a message preparation phase. Upon successful
completion, a FMH object is dynamically allocated with a FMH and a user
TCM having the TCB size equals to len, and a pointer to the FMH is returned
with the pointer *id set to the tmtcm_id of the user TCM. The returned FMH is
in the EDIT mode.

[0052]   In some embodiments the User TCM module is added directly
to the FMH to minimize the likelihood of it needing to be relocated in memory
at a later point in time. Since the user data can vary considerably both in size
and in content, placing it in this location reduces the chances of it having to be
moved again. Additional (non user data) TCM's, which are typically more
standard and smaller in size than User TCM's can then added on to the FMH.
Since the non-user data TCM's are smaller in size, the fact that they may need
to be relocated in memory at a later point in time is not as important.

Add TCM module (_tmfmsg_addtcm(EDIT))

```
TCBP _tmfmsg_addtcm(FMH *msg, unsigned short type, TM32U len,
unsigned short * id, TM32U flag)
```

[0053]   This function attaches a TCM module to the FMH. Upon successful completion, either a type TCM with len bytes of TCB is added to the FMH pointed to by the msg and the address of the new TCB is returned, or an existing "unused" TCM which has a TCB larger enough is re-used and the address of the existing TCB returned. The value of the tmtcm_id of the TCM is stored in a pointer *id. The _tminitbuf() functions is called to initialize the TCB. The inter-TCM sequence is not preserved.

[0054]   This function does not apply to the User TCM - as described above, in some embodiments the User TCM module is added directly to the FMH to minimize the likelihood of it being relocated in memory at a later point in time.

Count TCM (_tmfmsg_counttcm (EDIT))

```
int _tmfmsg_counttcm(FMH *msg, unsigned short type, TM32U flag)
```

[0055]   This function retrieves the number of TCMs having type. Upon successful completion, the total number of TCMs having type is returned. This function does not apply to the User TCM.

Find TCM (_tmfmsg_findtcm (EDIT))

```
TCBP _tmfmsg_findtcm(FMH *msg, unsigned short type, unsigned short
*id, TM32U flag)
```

[0056]   This function searches the TCM having the required type and the unique identification. Upon successful completion, the TCB address of either the TCM having type and also the identification or the first TCM having type if id is found to be NULL is returned. This function does not apply to the User TCM. The TCB is automatically decoded if needed and a postrecv function called if needed.

Find user TCM (_tmfmsg_findutcm (EDIT))

```
TCBP _tmfmsg_findutcm(FMH *msg,, char *type, char *subtype, TM32U
flag)
```

5   **[0057]**   This function searches the User TCM.   Upon successful completion, the TCB address of the User TCM is returned. This function only applies to the User TCM. The TCB is automatically decoded if needed and postrecv called if needed.

10   ### Resize User TCM (_tmfmsg_rszutcm(EDIT))

```
TCBP _tmfmsg_rszutcm(_TCADEF,FMH **msg, char *type, char *subtype,
TM32U len, TM32U flag)
```

[0058]   This function changes the size of the TCB portion of the User

15   TCM. Upon successful completion, the size of the TCB is changed to len bytes and the address of the re-sized TCB returned. The flag can be used for: TCM_NO_RESIZE - just to mark the new size and not do the reallocation when the new size is smaller than the old one; and TCM_RESET - to reset the old TCB when copying to a new TCM or reset part of the TCB when reducing

20   the TCB size.

Resize TCM (_tmfmsg_rsztcm(EDIT))

```
TCBP _tmfmsg_rsztcm(_TCADEF,FMH *msg, TCBP tcbp, TM32U len, TM32U
flag)
```

25

[0059]   This function changes the size of the TCB portion of the TCM. Upon successful completion, the size of the TCB is changed to len bytes and the address of the re-sized TCB returned. The flag can be used for: TCM_NO_RESIZE - just to mark the new size and not do the reallocation

30   when the new size is smaller than the old one; and TCM_RESET - to reset the

old TCB when copying to a new TCM or reset part of the TCB when reducing the TCB size. This function does not apply to the User TCM.

Delete TCM (_tmfmsg_deltcm(EDIT))

```
int _tmfmsg_deltcm(FMH *msg, TCBP tcbp, TM32U flag)
```

[0060]   This function invalidates a TCM from the FMH. Upon successful completion, the TCM is deleted or marked as free from the FMH object. The flag can be used for: TCM_NOT_USED - just to mark the TCM being "not used" but do not free it; and TCM_RESET - to initialize the TCB to zero. This function does not apply to the User TCM.

Decode TCM (_tmfmsg_dectcm(EDIT))

```
TCBP _tmfmsg_decutcm(FMH *msg, TCBP tcbp, TM32U flag)
```

[0061]   This function decodes a TCM. Upon successful completion, the address of the decoded TCB is returned. This function does not apply to the User TCM. The function _tmdecdec() is called if needed.

Decode User TCM (_tmfmsg_decutcm(EDIT))

```
TCBP _tmfmsg_dectcm(FMH **msg, char *type, char *subtype, TM32U flag)
```

[0062]   This function decodes the User TCM. Upon successful completion, the address of the decoded TCB is returned. This function only applies to the User TCM. The function _tmdecdec() is called if needed.

Presend (Edit) (_tmfmsg_presend(EDIT))

```
FMH *_tmfmsg_presend(FMH *msg, char *type, char *subtype, TM32U flag)
```

[0063] This function converts the FMH from the EDIT mode into the STORAGE mode. Upon successful completion, either the msg is returned or the address of a new FMH object is returned. The flag can be used for: ENCODE_TCM - encode all the TCMs (TCHs and TCBs). When the TCM is either encoded or unknown then the function _tmpresend() is not called.

Postsend (Edit, Storage) (_tmfmsg_postsend(EDIT, STORAGE))

```
FMH *_tmfmsg_postsend(FMH *msg, char *type, char *subtype, TM32U
flag)
```

[0064] This function converts the FMH from the after presend state back to the before presend state. When the TCM is either encoded or unknown, the function _tmpostsend() is not called.

Postreceive (Storage) (_tmfmsg_postrecv(STORAGE))

```
FMH *_tmfmsg_postrecv(FMH *msg, TM32U flag)
```

[0065] This function converts the FMH from the STORAGE mode into the EDIT mode. When the TCM type is unknown it will mark as unknown and the _tmpostrecv() function is not called, or if the TCM is encoded then _tmpostrecv() is not called. The User TCM is marked as unknown only. The inter-TCM sequence is not preserved.

Free Memory (_tmfmsg_free(EDIT, STORAGE))

```
int _tmfmsg_free(FMH *msg, char *type, char *subtype, TM32U flag)
```

[0066] This function frees up the memory space of the FMH. Upon successful completion, the FMH object is either marked as not used or freed. The flag can be used for: FMH_NOT_USED - just mark all the TCM as "not used" but do not free it; and FMH_RESET - initializes all the TCBs to zero.

Initialize Buffer (_tmuninitbuf())

[0067] The _tmuninitbuf() function is called for each TCM if it is not unknown, encoded or not in-used.

5   **[0068]** A detailed description of other API function calls that can be used with the invention, including those not specifically related to the FMH header structure, is given in the related U.S. Patent 5,265,250, incorporated by reference.

10   **Macros**

[0069] Macro functions can be used by the developer to retrieve useful information about the state of the FMH, TCM and TCB, and to assist in the messaging process. The macros shown here are given for purposes of illustration.

15

GETTCB (tcmp) - This macro will return the address of the TCB from the address of a TCM.

GETTCM (tcbp) - This macro returns the address of the TCM from the

20   address of a TCB.

ISTCBENC (tcmp) - This macro returns 1 if the TCB is encoded. Otherwise, it returns 0.

25   TCBSIZE (tcmp) - This macro returns the size of the TCB.

TCMID (tcmp) - This macro returns the unique identification of the TCM.

TCMTYPE (tcmp) - This macro returns the type of the TCM.

USERTCM(FMHp) - This macro returns the address of the User TCM from the address of the a FMH object.

FIRSTTCM (FMHp) - This macro returns the address of the first TCM from the
5   address of a FMH object.

NEXTTCM (tcmp) - This macro returns the address of the next TCM from the address of a given TCM.

10   ISTCMUSED(tcmp) - This macro returns 1 if the TCM is in-used. Otherwise, it returns 0.

Some error codes that may be generated for these functions are listed below:

15   TUX->_TUX_tperrno(TPEOS).

GP->_GP_Uunixerr(USBRK) - There is no available memory.

TUX->_TUX_tperrno(TPENOENT) - The type and subtype are not in the typed
20   buffer table (TCM Table).

TUX->_TUX_tperrno(TPEMATCH) - The tcmtype is currently used in the TCM table.

25   TUX->_TUX_tperrno(TPEINVAL) - The flag is not 0 or the tcmtype is part of the internal TCM table entries, or type is the User TCM type, or msg is NULL or has the wrong state, or flag is not 0, or TCB size is 0, or tcbp points to either an User TCM or a not in-used TCM or a TCM in wrong state, or msg is NULL or has the wrong state, or TCB size is 0.

TUX->_TUX_tperrno(TPESYSTEM) - The _tminitbuf() failed or _tmencdec()
failed or _tmpostrecv() failed, or _tmpresend() failed or _tmencdec() failed or
TCH encode failed, or TCH decode failed or _tmpostrecv() failed, or -
5   _tmuninitbuf() failed.

Message Flexibility and Integrity

[0070]   The invention allows for great flexibility and compatibility in
10   message handling throughout the release cycle. Since the end process (the
receiving application) can be designed or configured to look for, or to read,
only a particular portion of the FMH, or a particular subset of TCM's, the FMH
can be modified between releases (or between different system architectures)
to add extra functionality, while retaining compatibility with older releases (or
15   other system architectures). An older version of the application will simply
ignore the TCM's it is not configured to read, or will render the TCM as an
"unknown" type. The FMH is similarly capable of providing a message format
that is machine independent as well as release-independent.

[0071]   One of the potential problems with this approach is that an older
20   release of an application or process may unwittingly accept a message that
contains header information in a TCM it has rendered as "unknown", but which
in fact contains harmful code, for example a computer virus. To counter this,
embodiments of the invention are particularly well suited to ensuring the
integrity of the messaging process. This is particularly important for security-
25   related and secure applications. In accordance with this embodiment, the
messaging system includes a "fail on unknown" mechanism. A receiving
application or process can be configured so that, if it receives a TCM it does
not recognize, it fails, or rejects the message. A notification of the message
failure is then sent to the sending process. This mechanism can be used to

assure that the message is delivered with the integrity required by the sender. For example, the sender may specify a message with a format that includes a PKI (Public Key) security TCM. If the recipient is an older release that does not recognize the PKI TCM, then delivering that message would violate the

5   senders requirements - in this case not delivering the message is the safer option.

Industrial Applicability:

[0072]   The methods required for registration resolution of TCM type are

10 independent of the context of this application, and are dependent on the context of who will use the APIs and how to manage the "type" and "subtype" space among the users. The TCM type creation mechanism can, for example, use a simple pre-assigned type space for each user or perhaps a hashing process for some pre-defined type name formats that each user has to follow

15 in order to quarantine an unique type.

[0073]   The FMH provided by the invention changes the performance landscape in comparison with a fixed format message structure. The balance between the performance gain at the network bandwidth saving and the CPU processing when dealing with a smaller message header and the CPU

20 overhead when dealing with a non-fixed format message structure should minimize any performance penalty imposed on the actual application that uses the messaging system.

[0074]   The important issue of performance is to choose a design that can provide a flexible and generic framework that can support future growth

25 and that only adds cost at the incremental level with a minimum initial overhead. The FMH is one approach of using the modular concept to achieve that.

[0075]   The other application servers on the market had no prior solution other than the addition of unused fields which are not a viable long-term

solution. The TCM solution offered by the invention has several advantages as following:

[0076] New features can be added by adding new code and new TCM without changing the existing code.

5 **[0077]** Message content change is facilitated with great flexibility and efficiency. Old and new releases can be mixed in a single deployment without patching the old release.

[0078] The message footprint is reduced by allowing each message to tailor the FMH to its exact need instead of having a large common message

10 for all-purposes. Performance for forwarding message is increased by allowing a system component such as a server or intermediate recipient to examine a examining message piece by piece.

[0079] The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and

15 description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to the practitioner skilled in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand

20 the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.